CMCS 611-101
Advanced Computer Architecture

Lecture 9

Pipeline Implementation Challenges

October 5, 2009

www.csee.umbc.edu/~younis/CMSC611/CMSC611.htm

Lecture's Overview

- ***Previous Lecture:***
  - ➔ Control hazards
    - •Limiting the effect of control hazards via branch prediction and delay
    - •Static branch prediction techniques and performance
    - •Branch delay issues and performance
  - ➔ Exceptions handling
    - •Categorizing of exception based on types and handling requirements
    - •Issues of stopping and restarting instructions in a pipeline
    - •Precise exception handling and the conditions that enable it
    - •Instructions set effects on complicating pipeline design
- ***This Lecture***
  - ➔ Pipelining floating point operations
  - ➔ An example pipeline: MIPS R4000

Floating-Point Pipeline

- It is impractical to require that all MIPS floating point operations complete in one clock cycle (complex logic and/or very long clock cycle)
- A floating-point pipeline differs from integer instructions pipeline in:
  1. The EX cycle may be repeated as many times as need to complete the operation
  2. There may be multiple floating-point functional units
- A stall will occur if the instruction to be issued will either cause a structural or a data hazard
- No instruction is assumed to enter the EX stage as long the previous instruction has not finished it

Multi-cycle FP Pipeline

Integer unit

EX

FP/integer multiply

M1 M2 M3 M4 M5 M6 M7

IF ID

MEM WB

FP adder

A1 A2 A3 A4

FP/integer divider

DIV

7-stages pipelined FP multiple

Integer ALU

Non-pipelined DIV operation stalling the whole pipeline for 24 cycle

Pipelined FP addition with latency of 3 cycles

MULTD	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADDD		IF	ID	*A1*	A2	A3	**A4**	MEM	WB		
LD			IF	ID	*EX*	**MEM**	WB				
SD				IF	ID	EX	*MEM*	WB			

Example: *blue indicate where data is needed and red when result is available*

Multi-cycle FP Pipeline EX Phase

- *Example*: assume that floating point add, subtract and multiply can be performed in stages while integer and FP divide cannot
- **Latency**: the number of intervening cycles between an instruction that produces a result and an instruction that uses it
- Since most operations consume their operands at the beginning of the EX stage, latency is usually the number of stages in the EX an instruction uses
- Naturally long latency increases the frequency of RAW hazards
- **Initiation (Repeat) interval**: the number of cycles that must elapse between issuing two operations of a given type

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

Example of typical latency of floating point operations

FP Pipeline Challenges

Issues:

- Because the divide unit is not fully pipelined, structural hazards can occur
- Because the instructions have varying running times, the number of register writes required in a cycle can be larger than 1
- WAW hazards are possible, since instructions no longer reach WB in order
- WAR hazards are NOT possible, since register reads are still taking place during the ID stage
- Instructions can complete in a different order than they were issued, causing problems with exceptions
- Longer latency of operations makes stalls for RAW hazards more frequent

Instruction	Clock cycle number																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD F4, 0(R2)	IF	ID	EX	MEM	WB												
MULTD F0, F4, F6		IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADDD F2, F0, F8			IF	stall	ID	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	MEM	
SD 0(R2), F2					IF	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	MEM

Example of RAW hazard caused by the long latency

Write-Caused Structural Hazard

Instruction	Clock cycle number										
	1	2	3	4	5	6	7	8	9	10	11
MULTD F0, F4, F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
…		IF	ID	EX	MEM	WB					
…			IF	ID	EX	MEM	WB				
ADDD F2, F4, F6				IF	ID	A1	A2	A3	A4	MEM	WB
…					IF	ID	EX	MEM	WB		
…						IF	ID	EX	MEM	WB	
LD F2, 0(R2)							IF	ID	EX	MEM	WB

- At cycle 11, the MULTD, ADDD and LD instructions will try to write back causing structural hazard if there is only one write port
- Additional write ports are not cost effective since they are rarely used and it is better to detect and resolve the structural hazard
- Structural hazards can be detected at the ID stage and the instruction will be stalled to avoid a conflict with another at the WB
- Alternatively, structural hazards can be handled at the MEM or WB by rescheduling the usage of the write port

WAW Data Hazards

Instruction	Clock cycle number										
	1	2	3	4	5	6	7	8	9	10	11
MULTD F0, F4, F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
…		IF	ID	EX	MEM	WB					
…			IF	ID	EX	MEM	WB				
ADDD F2, F4, F6				IF	ID	A1	A2	A3	A4	MEM	WB
…					IF	ID	EX	MEM	WB		
LD F2, 0(R2)						IF	ID	EX	MEM	WB	
….							IF	ID	EX	MEM	WB

- WAW hazards can be corrected by either:
  - ❶ Stopping the latter instruction (LD in example) at the MEM until it is safe
  - ❷ Preventing the first instruction from overwriting the register
- Correcting WAW Hazards at cycle 11 is not problematic unless there is an instruction between ADDD and LD that read F2 causing RAW hazard
- WAW hazards can be detected at the ID stage and thus the first instruction can be converted to no-op
- Since WAW hazards are generally very rare, designers usually go with the simplest solution

Detecting Hazards

- Hazards can occur among FP instructions and among FP and integer instructions
- Using separate register files for integer and FP limits potential hazards to just FP load and store instructions
- Assuming all checks are to be performed in the ID phase, hazards can be detected through the following steps:
  1. Check for structural hazards:
     - Wait if the functional unit is busy (Divides in our case)
     - Make sure the register write port is available when needed
  2. Check for a RAW data hazard
     - Requires knowledge of latency and initiation interval to decide when to forward and when to stall
  3. Check for a WAW data hazard
     - Write completion has to be estimated at the ID stage to check with other instructions in the pipeline
- Data hazard detection and forwarding logic can be derived from values stored at the data stationary between the different stages

Maintaining Precise Exception

- Pipelining FP instructions can cause out-of-order completion since FP operations are different in length
- Although data hazard solutions prevent the effect of out-of-order instruction completion on the semantics, exception handling can be problematic
- Example:

DIVD	F0, F2, F4	ADDD and SUBD will complete before DIVD completes without causing any data hazards
ADDD	F10, F10, F8	
SUBD	F12, F12, F14	

What if an exception occurs while DIVD executes in a point of time after ADDD overwrites F10 !!

- There are four possible approaches to deal with FP exception handling:
  1. Settle for imprecise exceptions (restrict # active FP instructions)
     - IEEE floating point standard requires precise exceptions
     - Some supercomputers still uses this approach
     - Some machines offer slow precise and fast imprecise exceptions
  2. Buffer the results of all operations until previous instructions complete
     - Complex and expensive design (many comparators and large MUX)
     - History or future register file are practical implementations that allow for restoring original machine state in case of exception

Precise Exception Approaches (Cont.)

- Four possible approaches to deal with FP exception handling (cont.):
  - ❸ Allow imprecise exceptions and get the handler to clean up any miss
    - Simply save PC and some state about the interrupting instruction and all out-of-order completed instructions
    - The trap handler will consider the state modification caused by finished instructions and prepare machine to resume correctly
    - *Issues*: consider the following example

      Instruction_1: A long-running instr. that eventually get interrupted

      $\text{Instructions}_{2 \ldots (n-1)}$: A series of instructions that are not complete

      Instruction_n: An instruction that is finished

      ➔ The trap handler will have a complex job to deal with
    - The compiler can simplify the problem by grouping FP instructions so that the trap does not have to worry about unrelated instructions
  - ❹ Allow instruction issue to continue only if previous instruction are guaranteed to cause no exceptions:
    - Mainly applied in the execution phase
    - Used on MIPS R4000 and Intel Pentium

Performance of FP Pipeline

- Average number of stalls per instruction mainly affected by latency and number of cycles before results are used but not by instruction frequency (except for divide)
- Stalls are mainly caused by:
  - Latency of the divide instruction
  - RAW hazards
  - WAW hazards are possible but rare in practice

FP SPEC Benchmark	Add/Subtract	Compares	Multiply	Divide	Divide Structural
doduc	1.7	1.7	3.7	15.4	2.0
ear	1.6	2.0	2.5	12.4	0.0
hydro2d	2.3	2.5	3.2	0.4	0.0
mdljdp	2.1	1.2	2.9	24.5	0.0
su2cor	0.7	1.5	1.6	18.6	0.6

Number of stalls (axis: 0.0, 5.0, 10.0, 15.0, 20.0, 25.0)

Performance of FP Pipeline (Cont.)

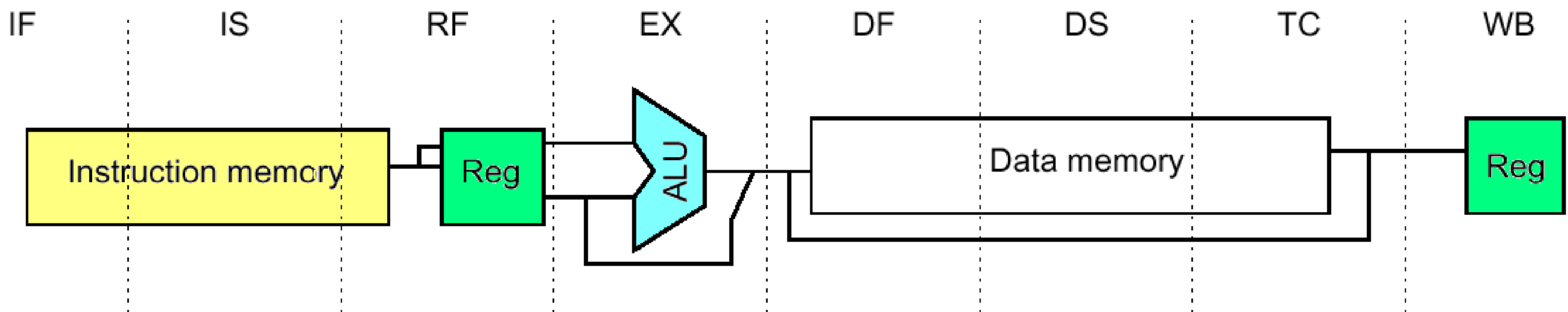
FP SPEC Benchmark	FP results stall	FP Compare stall	Branch/Load stalls	FP Structural
doduc	0.98	0.07	0.08	0.08
ear	0.52	0.09	0.07	0.00
hydro2d	0.54	0.22	0.04	0.00
mdljdp	0.88	0.10	0.03	0.00
su2cor	0.61	0.02	0.01	0.01

Number of stalls (x-axis: 0.00, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00)

- The total number of stalls/instr. range from 0.65 to 1.21 with average 0.87
- FP result stalls dominate in all cases with average of 0.71 (82% of all stalls)
- Compares generate an average of 0.1 stalls and are second largest source

MIPS 4000 Pipeline

- Implements the MIPS-3 64-bit instruction
- Uses 8 stages pipeline through pipelining instruction and data cache access
- Deeper pipeline allows for higher clock rate but increases load/branch delays



IF: First half of instr. fetch; PC selection, initiation of instruction cache access

IS: Second half of the instruction fetch completing instruction cache access

RF: Instr. Decode, register fetch, hazard checking and instr. cache hit detection

EX: ALU operations, effective address calculation, and condition evaluation

DF: Data fetch, first half of data cache access

DS: Second half of data fetch, completion of data cache access

TC: Tag check, determine whether the data cache access hit

WB: Write back for loads and register-register operations

Load Delays

Time (in clock cycles)

CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9 CC 10 CC 11

LW R1: Instruction memory, Reg, ALU, Data memory, Reg

Instruction 1: Instruction memory, Reg, ALU, Data memory, Reg

Instruction 2: Instruction memory, Reg, ALU, Data memory, Reg

ADD R2, R1: Instruction memory, Reg, ALU, Data memory, Reg

- ❑ Data value are available at the end of the DS cycle, cause a two cycle delay for load instructions
- ❑ Pipelined cache access increases the need for forwarding and complicates the forwarding logic
- ❑ A cache miss will stall the pipeline additional one (or more) cycles

Branching Delays

Time (in clock cycles)

CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8 CC9 CC10 CC11

BZ: Instruction memory, Reg, ALU, Data memory, Reg

Instruction 1: Instruction memory, Reg, ALU, Data memory, Reg

Instruction 2: Instruction memory, Reg, ALU, Data memory, Reg

Instruction 3: Instruction memory, Reg, ALU, Data memory, Reg

Target: Instruction memory, Reg, ALU, Data memory

- Branch conditions are computed during EX stage extending the basic branch delay to 3 cycles
- MIPS allows for a single-cycle delay branching and a predict-not-taken strategy for the remaining two branching delay cycles

MIPS 4000 FP Pipeline

- Three FP functional units: adder, multiplier and divider, with the adder logic serving as the final step of the a multiply and divide

Stage	Functional Unit	Description
A	FP adder	Mantissa ADD stage
D	FP Divider	Divide pipeline stage
E	FP Multiplier	Exception test stage
M	FP Multiplier	First stage of multiplier
N	FP Multiplier	Second stage of multiplier
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP numbers

- The FP functional unit can be thought as having eight different stages, combined in different order to execute various FP operations
- An instruction can use a stage zero or multiple times and in different orders

FP Instruction	Latency	Pipeline stages
Add/Subtract	4	U, S+A, A+R, R+S
Multiply	8	U, E+M, M, M, M, N, N+A, R
Divide	36	U, A, R, D^{27}, D+A, D+R, D+A, D+R, A, R
Compare	3	U, A, R

Performance of MIPS Pipeline

Base
Load stalls
Branch stalls
FP result stalls
FP structural stalls

Pipeline CPI

3.00
2.50
2.00
1.50
1.00
0.50
0.00

compress
eqntott
espresso
gcc
li
doduc
ear
hydro2d
mdljdp
su2cor

SPEC92 benchmark

Conclusion

- ***Summary***
  - ➔ Exceptions handling
    - Categorizing of exception based on types and handling requirements
    - Issues of stopping and restarting instructions in a pipeline
    - Precise exception handling and the conditions that enable it
    - Instructions set effects on complicating pipeline design
  - ➔ Pipelining floating point instruction
    - Multi-cycle operation of the pipeline execution phase
    - Hazard detection and resolution
    - Exception handling and FP pipeline performance
  - ➔ An example pipeline: MIPS R4000
- ***Next Lecture***
  - ➔ Instruction level parallelism
  - ➔ Dynamic instruction scheduling

Reading assignment includes sections A.5 & A.6, in the textbook